ANN queries: covering Voronoi diagram with hyperboxes

Rajasekhar Inkulu * Sanjiv Kapoor †

Abstract

Given a set S of n points in d-dimensional Euclidean metric space X and a small positive real number ϵ, we present an algorithm to preprocess S and answer queries that require finding a set $S' \subseteq S$ of ϵ-approximate nearest neighbors (ANNs) to a given query point $q \in X$. The following are the characteristics of points belonging to set S':

$\forall s \in S'$, $\exists$ a point $p \in X$ such that $|pq| \leq \epsilon$ and the nearest neighbor of p is s, and
$\exists$ a $s' \in S'$ such that s' is a nearest neighbor of q.

During the preprocessing phase, from the Voronoi diagram of S we construct a set of box trees of size $O(4^d \frac{V}{\delta}(\frac{\pi}{\epsilon})^{d-1})$ which facilitate in querying ANNs of any input query point in $O(\frac{1}{d} lg \frac{V}{\delta} + (\frac{\pi}{\epsilon})^{d-1})$ time. Here δ equals to $(\frac{\epsilon}{2\sqrt{d}})^d$, and V is the volume of a large bounding box that contains all the points of set S. The average case cardinality of S' is shown to rely on S and ϵ.

1 Introduction

Nearest neighbor searching has applications in knowledge discovery, data mining, pattern classification, machine learning, data compression, multimedia databases, and document retrieval.

From the perspective of worst-case performance, an ideal solution for the nearest neighbor search would be to preprocess the points in $O(n \lg n)$ time, into a data structure requiring $O(n)$ space so that any query can be answered in $O(\lg n)$ time. In R^1, this is possible by sorting S, and then using binary search to answer queries. In R^2, this is possible by computing the Voronoi diagram for S and then using an algorithm for planar point location to find the cell containing the query point. However, in higher dimensions, the worst-case complexity of the Voronoi diagram grows as $O(n^{\lceil d/2 \rceil})$. Specifically, no known method achieves the simultaneous goals of roughly linear space and logarithmic query time in higher dimensional spaces. This is the primary reason in resorting to approximation algorithms. Given a set S of n points, a point $p \in S$ is an ϵ-approximate nearest neighbor (ANN) of a point $q \in R^d$, if the distance between q and p is at most ϵ-times the Euclidean distance between q and its nearest neighbor in S. However, even in approximation schemes, usually the space, time, and query time complexities grow exponential in ϵ.

Netanyahu et al., [2] presents an algorithm to find an ANN of a query point q in time $O(c \lg n)$ (c relies on d and ϵ), with a data structure of size $O(dn)$ constructed in $O(dn \lg n)$ time. Their data structure, BBD tree, is based on the hierarchical decomposition of space: subdivide space into a collection of cells, each of which is either a d-dimensional rectangle or a set-theoretic difference of two rectangles, one enclosed within another. The BBD-tree is based on a spatial decomposition that achieves both exponential cardinality decrease (like kd-trees) and the geometric size reduction (like quadtrees) as one descends the tree. After finding the leaf cell containing the query point q in $O(\lg n)$ time by a simple descend through the tree, one can find q's approximate nearest neighbors: enumerate the leaf cells in increasing order of distance from q until distance to the point p associated with a cell exceeds $dist(p', q)/(1 + \epsilon)$ where p' is the closest point to q in the output

*Department of Computer Science, Indian Institute of Technology Guwahati, India, rinkulu@iitg.ac.in
†Department of Computer Science, Illinois Institute of Technology, Chicago, USA, kapoor@iit.edu

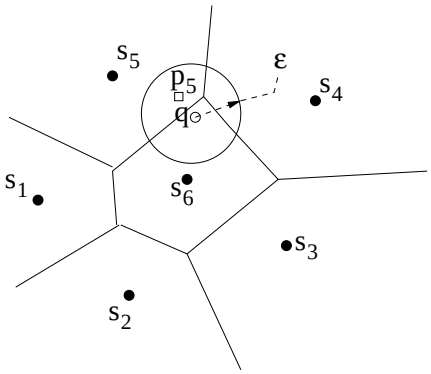


Figure 1: Definition of ϵ-approximate nearest neighbors

so far. The algorithm from Arya et al., [3] minimizes the number and complexity of cells by constructing an approximate Voronoi diagram of S during the preprocessing phase. This yields a data structure of $O(n\gamma^{d-1} \lg \gamma)$ space that can answer ϵ-ANN queries in time $O(\lg (n\gamma) + 1/(\epsilon\gamma)^{(d-1)/2})$ for a space-query time trade-off parameter γ, where $2 \leq \gamma \leq 1/\epsilon$. The principal reduction in space arises from a form of deterministic sampling over BBD-tree. The approach also creates cells more economically using well-separated pair decomposition (WSPD) of the points by generating them along the bisector between well-separated pairs. In the case $\gamma = 1/\epsilon$, the additional $\lg \gamma$ factor in space complexity can be avoided to have a data structure that answers queries in $O(\lg (n/\epsilon))$ time with $O(n/\epsilon^{d-1})$ space (refer Arya et al., [1] and Chan [4]). Har-Peled [5] and Sabharwal et al., [6] suggests an approach to reduce the query version of ANN problem to point location in balls.

In this paper, we are interested in querying for ϵ-ANNs but the definition of ϵ-ANN is bit varied from the literature. To our knowledge, this seems to be the first paper to give an algorithm for the following variant definition of ϵ-ANN. For a query point q, the output of our algorithm is a set $S' \subseteq S$ wherein: $\forall s \in S'$, $\exists$ a point p in R^d such that $|pq| \leq \epsilon$ and the nearest neighbor of p is s, and $\exists$ a $s' \in S'$ such that s' is a nearest neighbor of q.

As shown in Fig. 1, let $\{s_1, s_2, \ldots, s_6\}$ be the set S. Consider a query point q that belongs to the Voronoi region of s_6. Suppose $S' = \{s_5, s_6\}$. The site s_5 is in S' as there exists a point p_5 such that $|qp_5| \leq \epsilon$ and s_5 is the nearest neighbor of p_5. Similarly, the site s_6 is in S' becuase of $|qq| \leq \epsilon$ and s_6 is the nearest neighbor of q. Consistent with the definition of ϵ-ANN, not all sites that correspond to regions intersecting with the ϵ-radius ball centered at q are in S' (for example, $s_4 \notin S'$).

We first construct the Voronoi diagram VD of S. We do this by space partitioning VD (contained in a large bounding hyperbox) with smaller hyperboxes and organizing hyperboxes into a box tree data structure, termed as *main box tree*. We also cover specific regions of VD with hyperboxes whose bounding planes are not parallel to coordinate-planes and construct a set of corresponding box tree data structures, each of which is termed as an *auxiliary box tree*. The advantage is that the query phase does not require data structures corresponding to VD. It only requires main and auxiliary box tree data structures. Although our algorithm does not improve time or space bounds of previous algorithms, it seems to provide another direction (detailed in the Section 5) for approaching query version of ANN finding problem. This is the main motivation in listing this result.

2 Algorithm

Our algorithm first constructs the Voronoi diagram VD corresponding to the input set S of points in d-dimensional Euclidean metric space. Before deleting data structures corresponding to VD, we construct a set of box trees from VD. We enclose a sufficiently large space of VD in a bounding hyperbox BB so that every site in S is an ANN to any point exterior to BB. We spatially

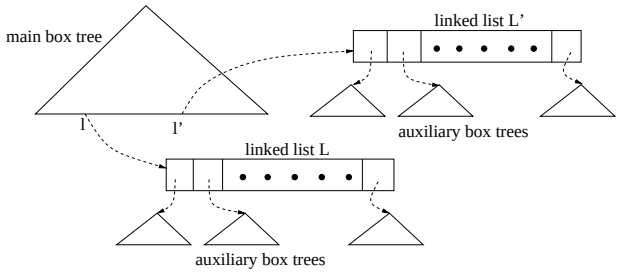


Figure 2: Data structures at the end of preprocessing phase

partition BB into hyperboxes of volume δ (a real number that relies on ϵ), and with each such hyperbox HB we associate sites corresponding to Voronoi cells with which HB intersects. We organize these hyperboxes into a box tree, termed as the *main box tree* (refer Fig. 2). Whenever the hyperbox HB stored at any leaf of main box tree is associated with multiple sites, we cover the region enclosed by HB with multiple hyperboxes, termed as *auxiliary hyperboxes*. Every such hyperbox is obtained by rotating HB w.r.t. its center by a distinct discrete angle. By partitioning the region enclosed by each of these auxiliary hyperboxes, we construct a box tree, termed as an *auxiliary box tree* (refer Fig. 2). Like in the main box tree, leaf node l of every auxiliary box tree is associated with Voronoi sites corresponding to Voronoi cells that intersect with the hyperbox at l.

Each internal/leaf node of main/auxiliary box tree implicitly associated with the hyperbox that corresponds to union of hyperboxes that are associated with its children. Also, every internal node v has information necessary to locate q in the hyperbox associated with a specific child of v. Given a query point q for which the user is interested in finding ANNs, we descend through the height of the main box tree in finding the leaf node l whose hyperbox HB contains q. If HB is associated with one and only one Voronoi site, we have determined the exact neighbor of q. Otherwise, similar to traversing the main box tree, we traverse each of the auxiliary box trees associated with leaf node l so as to confine the cardinality of set S' output by the algorithm, as detailed below. Essentially, during the query phase we rely on main and auxiliary box trees rather than on data structures that store Voronoi structure.

2.1 Preprocessing phase

The preprocessing phase starts by constructing a Voronoi diagram.

The box tree data structure is a trivial generalization of quadtree data structure to higher dimensions. It is a rooted tree in which every internal node has 2^d children. The root node of the main box tree corresponds to a hyperbox that is the union of all the hyperboxes into which BB is partitioned. The children of any internal node v are obtained by subdividing hyperbox HB corresponding to v with the hyperplanes parallel to coordinate planes, which bisect each dimension of HB. In other words, every internal node v is partitioned into hyperboxes, and each of them is implicitly stored at a distinct child of v. Note that since the box is symmetric the direction of rotation is not important. Each leaf node l of the main box tree stores the sites corresponding to Voronoi cells that intersect with the hyperbox at l as satellite data. A point in S belongs to which half-space defined by a hyperplane is decided by trivial means. This elementary operation assists in associating sites with the leaf nodes.

The recursive definition of the box tree immediately translates into a recursive algorithm: partition the hyperbox associated with a node v into 2^d hyperboxes using hyperplanes parallel to coordinate-planes. The recursion stops at a leaf node l of the main box tree whenever the volume of hyperbox HB at l is less than a real number δ (which is defined in terms of ϵ).

Consider the case in which the hyperbox HB at a leaf node l is associated with multiple sites but the volume of HB is less than δ. In this case, we associate a linked list of auxiliary box trees

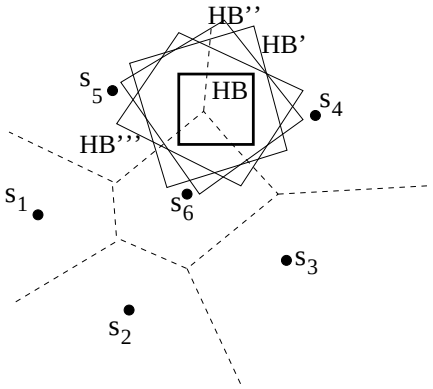


Figure 3: Auxiliary hyperboxes corresponding to roots of auxiliary box trees

to leaf node l. The root of each of these auxiliary box trees correspond to an auxiliary hyperbox which is constructed by rotating HB by a solid angle ϵ w.r.t. its center. However, since a query point that belongs to HB does not necessarily contain in any such auxiliary hyperbox, we double the size of HB before orienting it.

Let HB'_1 be the hyperbox that is obtained by scaling HB by two w.r.t. its center (refer Fig. 3). The root of the first auxiliary box tree in the linked list is associated with a hyperbox HB''_1, that is obtained by rotating HB'_1 w.r.t its center by ϵ solid angle. The next auxiliary box tree in the linked list is obtained by rotating HB''_1 by ϵ etc., so that no two of the auxiliary hyperboxes corresponding to two distinct auxiliary box trees are oriented with the same angle. In other words, the length of the linked list associated with l is $(\frac{\pi}{\epsilon})^{d-1}$. Each of the auxiliary box trees are defined similarly as in main box tree: partition the hyperbox associated with a node v into 2^d hyperboxes using hyperplanes parallel to coordinate axes. The recursion stops at a leaf node l whenever the volume of a hyperbox corresponding to a node is less than δ.

2.2 Query phase

Given a query point q in d-dimensional Euclidean space, we intend to find its ANNs. First, we find the leaf node in main box tree that contains q. This is accomplished by traversing one node at each level of the main box tree. The satellite data associated with internal nodes assists in branching to a particular child of v. The same could be accomplished in $O(1)$ time using hashing i.e., by maintaining a hash table rather than a box tree. However, to maintain the consistency between primary and secondary level data structures, we organize primary level data structure as a box tree.

Let q belongs to a hyperbox HB associated with a leaf node l of the main box tree. If HB is associated with a single site s, then it is immediate to conclude that the exact neighbor of q is s. Otherwise, we traverse through the linked list L referred by node l. As mentioned earlier, each node of the linked list refers to an auxiliary box tree. Similar to the traversal of main box tree, we traverse each of the auxiliary box trees referred by L, say $HB'_1, HB'_2, \ldots, HB'_r$. Let $R \subset \{HB, HB'_1, \ldots HB'_r\}$ be the set that comprises of hyperboxes that contain q. Also, let $S_1, \ldots, S_{|R|}$ be the sets of sites respectively associated with hyperboxes in R. The algorithm returns the set $S' = S_1 \cap S_2 \cap \ldots \cap S_{|R|}$ as the set of ANNs to q.

3 Correctness

From the Analysis mentioned in Section 4, it is immediate that in both the preprocessing phase and in the query phase algorithm always terminates. We have chosen the size of bounding box BB

so that the algorithm is correct for all the query points exterior to BB.

Let q belongs to a hyperbox HB that corresponds to a leaf node l of the main box tree. If HB is associated with a single site, according to the definition of Voronoi diagram, the algorithm outputs the nearest neighbor of q. Otherwise, the proof of correctness follows.

We are ensuring that the hyperbox corresponding to the root of every auxiliary box tree contains q by doubling the size of HB. The intersection of all the auxiliary hyperboxes associated with any leaf node l together yield a (approximate) hypersphere B whose boundary consists of hyperboxes of size ϵ in R^{d-1}. Given the volume of the byperbox is δ, the length of the main diagonal of a hyperbox is $\sqrt{d}\delta^{1/d}$. Hence, the diameter of hyperball B is less than or equal to $2\sqrt{d}\delta^{1/d}$.

Let S' be the set of sites whose Voronoi regions intersect with the hyperball B. Suppose q lies in B. In this case, though the output of the query algorithm does not have any site from the set $S-S'$, no auxiliary box tree eliminates any site in S'. By choosing the diameter of B as ϵ, every site in S' is an ANN of q. Hence, we choose δ as $(\frac{\epsilon}{2\sqrt{d}})^d$. Consider the other case in which q does not lie in B. In this case, set intersection of sites associated to multiple auxiliary hyperboxes together define the ANNs of q. Since the main diagonal lengths of every hyperbox that is implicitly stored at any leaf node is upper bounded by ϵ, the correctness is ensured again.

4 Analysis

Theorem 4.1 *Assuming uniform distribution of S in R^d Euclidean metric space, the expected cardinality of set S' is $(\frac{\epsilon}{2r})^d|S|$. Here, r is the radius of the smallest hyperball that contains S.*

Proof: First note that the hyperball of radius r has volume $C_d r^d$, where $C_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$. Due to uniform distribution of S, the expected number of sites per unit volume are $\frac{|S|}{C_d r^d}$. Since the diameter of the hyperball B is ϵ, the number of sites in B are $(C_d(\frac{\epsilon}{2})^d)(\frac{|S|}{C_d r^d})$. □

Lemma 4.1 *The depth of the main box tree is at most $\frac{1}{d}\lg\frac{V}{\delta}+1$, where V is the volume of the initial bounding box and δ is the volume tolerance of the smallest possible box.*

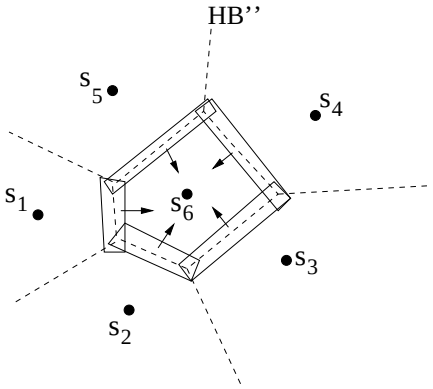
Proof: The box volume goes down by 2^d for every level, therefore the volume of the box at depth i is $\frac{V}{2^{di}}$. We know that the volume δ is possible at one less than the maximum depth. Therefore, the smallest possible box volume is, $\frac{\delta}{2^d}$. Hence, for any hyperbox in the main box tree $\frac{V}{2^{di}} \geq \frac{\delta}{2^d} \Rightarrow i \leq \frac{1}{d}\lg\frac{V}{\delta}+1$. □

Lemma 4.2 *The maximum possible number of nodes in the main box tree are $\frac{2^{2d}}{2^d-1}\frac{V}{\delta} - \frac{1}{2^d-1}$*

Proof: If we consider the complete box tree, the number of boxes in the box tree at the ith level are 2^{di}. Therefore, the total number of nodes in the main box tree are $1+2^d+2^{2d}+...+(\frac{1}{d}\lg\frac{V}{\delta}+2$ terms), summing over this geometric progression yields the result. □

Lemma 4.3 *The maximum depth of any auxiliary box tree is $O(1)$.*

Proof: We know that the volume of any hyperbox HB at a leaf node of main box tree is δ. Since we scale HB by two, the volume of the resultant hyperbox at the root of auxiliary box tree is 2δ. Then from Lemma 4.1, the depth of the auxiliary box tree is at most $\frac{1}{d}\lg\frac{2\delta}{\delta}+1$, which is $O(1)$. □



Arrows indicate the direction of expansion of hyperboxes into Voronoi cell so that they together cover that cell.

Figure 4: Suggested approach

Lemma 4.4 *The maximum number of nodes in any auxiliary box tree are* $\frac{2^{2d+1}-1}{2^d-1}$.

Proof: This is immediate by substituting $V = 2\delta$ in the statment of Lemma 4.2. □

Theorem 4.2 *Given the Voronoi diagram of S, the space complexity in constructing all box trees together is* $O(4^d \frac{V}{\delta}(\frac{\pi}{\epsilon})^{d-1})$.

Proof: The main box tree is a complete tree and the number of leaf nodes are $2^d \frac{V}{\delta}$. Also, each leaf may have a list of length $(\frac{\pi}{\epsilon})^{d-1}$. From Lemma 4.4, the space complexity is as stated. □

Theorem 4.3 *The preprocessing time in constructing all the box trees is* $O(4^d \frac{V}{\delta}(\frac{\pi}{\epsilon})^{d-1} + (dn)(\frac{\pi}{\epsilon})^{d-1} + \frac{1}{d}\lg(\frac{V}{\delta}) + n^{\lceil \frac{d}{2} \rceil})$.

Proof: Since only constant time is spent in creating each node of every box tree, the time complexity is same as the space complexity except that we need to consider the following. The cost in associating sites to main box tree leaves involve dn point location searches per each level of the main box tree. The same is applicable to auxiliary box trees. Including the time complexity of constructing the Voronoi diagram of S yields the stated. □

Theorem 4.4 *The query time complexity is* $O(\frac{1}{d}\lg\frac{V}{\delta} + (\frac{\pi}{\epsilon})^{d-1})$.

Proof: From Lemma 4.1, the worst-case depth of the main box tree is $\frac{1}{d}\lg\frac{V}{\delta} + 1$. The worst-case length of any list at a leaf node of the main box tree is $(\frac{\pi}{\epsilon})^{d-1}$. From Lemma 4.3, the worst-case height of the auxiliary box tree is O(1). Therefore, the worst-case query time is $\frac{1}{d} lg \frac{V}{\delta} + 1 + (\frac{\pi}{\epsilon})^{d-1} O(1)$. □

Note that each occurrence of δ in the above Analysis can be expressed in terms of ϵ, as $\delta = (\frac{\epsilon}{2\sqrt{d}})^d$.

5 Conclusions

This paper proposed an algorithm that builds data structures of size $O(4^d \frac{V}{\delta}(\frac{\pi}{\epsilon})^{d-1})$ during the preprocessing phase which facilitate in querying ANNs of an input query point in $O(\frac{1}{d}\lg\frac{V}{\delta} + (\frac{\pi}{\epsilon})^{d-1})$ time. Here δ equals to $(\frac{\epsilon}{2\sqrt{d}})^d$, and V is the volume of the large bounding box that contains set S of

points. The expected cardinality of output set S' corresponding to a given query point is shown to rely on S and ϵ. Although this algorithm does not improve either space or query-time complexity of existing algorithms, it suggests an approach to explore the problem under the following setting: Independently cover each Voronoi cell with $O(\frac{1}{\epsilon^d})$ hyperboxes (refer Fig. 4); this leads to a space complexity of $O(\frac{n}{\epsilon^d})$ (given that there are at most n Voronoi cells). Organize all the hyperboxes covering the entire Voronoi diagram into a data structure, similar to the one proposed herewith, to achieve query time efficiency.

References

[1] S. Arya and T. Malamatos Linear-size approximate Voronoi diagrams. In *Proc. 13th ACM-SIAM Symp. Discrete Algorithms (SODA)*, pages 147–155, 2002.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, A. Y. Silverman, and R. Wu An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[3] S. Arya, and T. Malamatos, D. M. Mount Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57: 1–54, 2009

[4] T. M. Chan Approximate nearest neighbor queries revisited. *Discrete Comput. Geom.*, 20:359–373, 1998.

[5] S. Har-Peled A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Ann. Symp. Foundations of Computer Science (FOCS)*, pages 94–103, 2001.

[6] Y. Sabharwal, S. Sen, and N. Sharma Nearest neighbors search using point location in balls with applications to approximate Voronoi decompositions. *J. Comput. Sys. Sci.*, 72:955–977, 2006.